VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Business metody sériového vývoje systémů pro podporu Multilevel marketingu

Methods for Product Line Development of System for Multilevel Marketing

2013 Bc. Antonín Rykalský

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Antonín Rykalský**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Business metody sériového vývoje systémů pro podporu Multilevel marketingu
Methods for the Product Line Development of System for Multilevel Marketing

Zásady pro vypracování:

Cílem práce je popsat metody vývoje netriviálních informačních systémů. Primárně se zaměřuje na obchodní i technický postup zpracování projektů. Obchodní záležitosti budou zpracovány formou business case. Práce bude demonstrována na systému pro multilevel marketing.

Práce bude obsahovat a zabývat se zejména:
1. Optimalizací aplikace pro webový provoz - doporučené postupy a ukázka na aplikaci.
2. Automatickým testování systému
3. Modulárností aplikace - způsoby řešení a jejich provedení - ukázka na aplikaci.
4. Eliminací výskytu bezpečnostních chyb a jejich zneužití - bezpečnostní rizika, která mohou nastat a jak se jim vyvarovat.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Svatopluk Štolfa, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě 7. května 2013 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 7. května 2013 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla. Poděkování patří mému vedoucímu diplomové práce panu inženýrovi Svatoplukovi Štolfovi, Ph.D. za odborné vedení a spoustu cenných rad a taktéž všem svým klientům, kteří se mnou měli dostatek trpělivosti.

Abstrakt

Diplomová práce "Business metody sériového vývoje systémů pro podporu Multilevel marketingu" popisuje metodiky a praktiky, které lze efektivně využít při tvorbě tohoto typu systémů. Vybranou metodiku autor přiblíží a zhodnotí její přínosy a zápory. Autor dále popisuje jak využít navrženou modularitu pro lepší znuvupoužitelnost a další vývoj modulů. V dalším bodě práce obsahuje popis, jak můžeme aplikaci optimalizovat, aby byla schopna fungovat přes webové rozhraní. Přibližuje zpracování obchodního případu a jak jej využít při rozhodování o vstupu do projektu. Práce dále řeší zefektivnění vyhledávání vad v software při vývoji a údržbě systému pomocí automatického testování. A popisuje analýzu rizik, které mohou být v projektu hrozbou.

Klíčová slova: metodologie vývoje software, optimalizace pro webový provoz, business case, automatické testování, analýza rizik, modularita

Abstract

The thesis "Methods for Product Line Development of System for Multilevel Marketing" describes the methodology and practices that can be effectively used to create this type of systems. Author approaches the selected methodology and practice their benefits and drawbacks. The author then describes how to use designed modularity to increase further reusability of development modules. The next section contains the description of how we can optimize the application to be able to work through a web interface. It shows the business case process and how to use it to decide about joining the project. The work also addresses the efficiency of search of defects in software development and maintenance of the system using automated testing. A risk analysis describes that the project can be a threat.

Keywords: software methodology, optimalization for web interface, business case, automatized testing, risk analysis, modularity

Seznam použitých zkratek a symbolů

BVT	–	Verifikační testy sestavení (angl. Build Verification Tests)
CI	–	Průběžná integrace (angl. Continuous Integration)
DOM	–	Objektový model dokumentu neboli objektově orientovaná reprezentace XML nebo HTML dokumentu (angl. Document Object Model)
HTTP	–	Hypertext Transfer Protocol
HTTPS	–	Hypertext Transfer Protocol Secured
MVC	–	Model View Controller; třívrstvá architektura software
presenter	–	ekvivalent slova Controller z MVC; Nette framework využívá využívá toto názvosloví
QA	–	Quality assurance
SRS	–	Specifikace požadavků na systém (angl. System requirement specification)
TDD	–	Vývoj řízený testy (angl. Test Driven Development)
UI	–	User Interface
XP	–	eXtreme programming
XSS	–	Cross Site Scripting

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Informační systémy pro podporu podnikání mají na softwarovém trhu stálou poptávku. Systémy pro *multilevel marketing a věrnostní programy*[1] tvoří menší poptávanou podmnožinu na trhu. Tím méně firem se na tento segment zaměřuje.

MLM software má svá specifika. Tak jako internetové obchody budou vždy mít nákupní košík, objednávkový proces, seznamy a detaily produktů, tak MLM software vždy bude mít zařazení uživatele do grafové struktury, bodová/peněžní či jiná konta, výběry peněz z konta, distribuce administrativy do grafové struktury uživatelů aj.

V předchozím odstavci jsem chtěl poukázat na podobnost systémů. Na druhou stranu má každý systém své odlišnosti, unikátní proces tvorby a jiné zainteresované osoby.

1.1 Cíle práce

V jedné větě by cíl této práce mohl být popsán: *jak efektivně vyvíjet netriviální informační systémy profesionálně.*

Zpracování obchodních příležitostí
První kapitola nazvaná zpracování obchodních příležitostí se bude věnovat, zpracování potencionálních obchodních příležitostí. Jakým způsobem specifikovat zakázku, tak aby zadání bylo jednoznačné a dalo se na něm stavět v dalších částech životního cyklu vývoje. Dále bude věnována složitosti a rizikům zpracování obchodních příležitostí.

Dále jsem v zadání práce jako čtyři hlavní cíle vytyčil tyto body:

Optimalizace aplikace pro webový provoz
Zde bude popsán návrh aplikace takovým způsobem, aby zvládla chod v reálném čase na webovém serveru. Tomuto tématu je věnována kapitola 3.

Automatické testování aplikace
Bez zajištění testování, nelze vyvíjet kvalitní software. Každý dobrý programátor ví, že by měl testovat svůj kód. Problémem je, že ne každý to dělá. Kapitola 4 se zabývá, jakým způsobem provádět testování automaticky, nekonvenčně, ale efektivně a jak si s ním usnadnit vývoj.

Modulárnost aplikace
Kapitola 5 se zabývá modulárností aplikace, konkrétně jak dosáhnout toho, abychom mohli specifika aplikace znovu výhodně používat v jiných projektech. Na druhou stranu, ne vždy je čas na kvalitní refactoring a může se stát, že použijeme chybný kód, který opravíme až v dalším projektu. Budeme se věnovat tomu, jakým způsobem distribuovat opravený kód i do ostatních aplikací, kde je tento kód použit.

[1] dále v textu budu používat jen pojem MLM software

Eliminaci výskytu bezpečnostních chyb
A posledním z hlavních cílů této práce je popsat bezpečnostní hrozby a jejich zneužití, jež najdete v kapitole 6. Touto kapitolou se musíme zabývat minimálně z důvodu, že systém počítá s penězi.

1.2 Obecně o MLM

Multilevel marketing je forma přímého prodeje prostřednictvím distribuční sítě nezávislých prodejců. Tito prodejci, kromě prodeje produktů také vyhledávají nové prodejce, čímž si budují svou distribuční síť. Jejich finanční příjmy jsou dány aktivitou jejich prodeje a aktivitou prodeje v jejich distribuční síti. [1] str.277

Podstatou MLM je zkrácení distribučního řetězce, kdy distribuční společnost nebo výrobce, vytváří podmínky pro samostatné podnikání svých nezávislých přímých prodejců. Předpokladem je, že tato distribuční síť, by měla být ekonomicky výhodnější než v případě koncepčního podnikání. [2] str.12

Společnost provozující MLM tedy vytváří svůj systém, jakým se budou distribuovat finanční odměny z prodejů produktů do distribuční sítě, formou provizí. Existují typické provize, které tyto společnosti využívají, které jsou popsány v dalších podkapitolách. Tyto standardní provizní systémy jsou často upravovány na míru podle produktu, který společnost distribuuje, a jsou k nim často přidávány nové vlastnosti.

1.2.1 Kariérní provize

Kariérní provize jsou uživatelům přidělovány na základě bodů a koeficientu. Koeficient je zde určen pro každou z kariérních skupin, do které se klient kvalifikuje svou aktivitou v systému. Body se uživatelům načítají body z distribuce produktu. Podle počtu těchto bodů se pak po skončení účetního období uživatel kvalifikuje do jednoho z předem stanovených intervalů - kariér - ve kterém je koeficientem určen přepočet bodového konta na peníze.

Specifikem těchto systémů je, že provizi z distribuce produktu, získává pouze první uživatel v každé z kariér z každého jednotlivého nákupu ve struktuře prodejců směrem ke kořeni stromové struktury.

V těchto systémech ještě existují dvě standardní podskupiny.

Načítací systémy bych označil ty, kde se konta uživatelů, podle kterých se kvalifikují do kariérní tabulky, střádají přes všechna účetní období. Přepočet na peníze se pak děje z pouze z bodů získaných v aktuálním období.

Padací systémy jsou dány vlastností, kde se konta uživatelů, podle kterých se kvalifikují do kariérní tabulky, nulují před každým účetním obdobím. Tím je zajištěna distribuce odměn ze sítě pouze aktivním uživatelům.

úroveň	procento
1.	20%
2.	10%
3.	10%
4.	5%
5.	5%

Tabulka 1: Příklad kariérní tabulky využívající fixní provize

1.2.2 Fixní provize

Tento druh provizí je vypočítáván, procentuálně-fixně dle zadaných parametrů. Například, je-li marže z prodeje produktu je 2000 Kč, je tato marže se rozdělena mezi klienty-doporučitele podle provizní tabulky, jako je například uvedena v tabulce 1. Zbytek marže je zisk firmy.

1.2.3 Jiné typy

Variací těchto systémů existují spousty a cílem této práce není jejich vyčerpávající popis. Například nově se objevují i tzv. *Binární systémy* kde se mění grafová struktura. A i sami zákazníci si vymýšlejí stále nové konstrukce.

V případě zájmu o hlubší poznání těchto systémů bych čtenáře odkázal na práci Multilevel marketing ve finančních službách, viz [3].

1.3 Předpoklady technického řešení

Předpokladem tvorby této práce je tvorba nad skriptovacím programovacím jazykem PHP 5.3 nebo vyššími verzemi, zejména z důvodů cenově dostupného hostingu a velkého množství schopných programátorů pro použití v praxi.
Dalším předpokladem je že se bude jednat o plně webovou aplikaci. Toto řešení má své omezení[3], nicméně pro jednoduchost se v této práci omezíme pouze na tento způsob.

[3] například omezený výpočetní výkon nebo bezpečnost dat

2 Obchodní příležitostí

2.1 Zpracování obchodních příležitostí

V této kapitole, budou popsány kroky vedoucí k vymezení požadavků na aplikaci, následné transformaci těchto požadavků na S.M.A.R.T[4] úkoly a aplikace softwarového procesu na tyto úkoly.

2.1.1 Sběr a vymezení požadavků na aplikaci

"Zákazník neví, co chce"[5]. Jakmile přijde situace, že osoba zodpovědná za sběr požadavků - analytik/vedoucí projektu/doménový expert - sedne k jednomu stolu s klientem, který je připraven začít řešit jaký systém potřebuje, analytik musí převzít kontrolu nad situací a začít požadavky sbírat systematicky.

V této situaci je dobré mít předpřipravenu určitou kostru (angl. *skeleton*) celé konverzace. Této kostry je vhodné se držet a pomocí tzv.*soft skills* získat od zákazníka, co skutečně potřebuje. Aktuální kostru najdete v elektonické příloze E.

Pomocí této kostry a vhodně mířených otázek, zaříkáme tvořit specifikaci požadavků na systém (angl. System requirement specification, SRS).

2.1.2 Zpracování specifikace požadavků

Pro zpracování požadavků byla vytvořena aplikace ***spec***, která je navržena pro zařazení požadavku do kontextu systému.

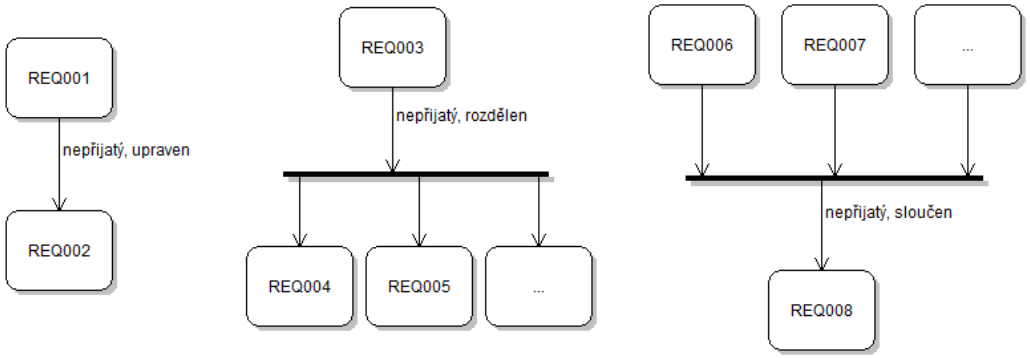
Aplikaci *spec* naleznete v elektronické příloze A.

Myšlenky použité pro sběr:

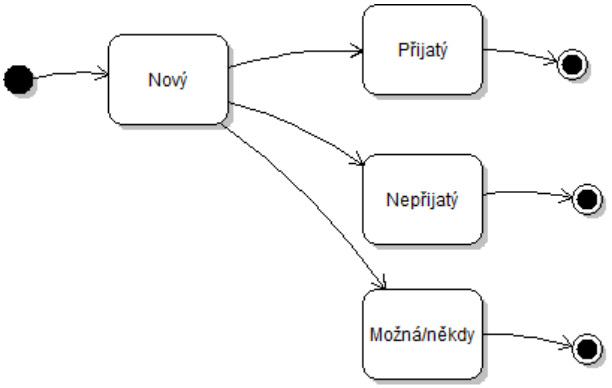
1. Požadavky zpracováváme od obecnějších ke konkrétnějším
2. Požadavkům přiřazujeme stavy
    - Nový požadavek
    - Přijatý požadavek
    - Nepřijatý, upraven
    - Nepřijatý, rozdělen
    - Nepřijatý, sloučen s jiným

[4]Specific - konkrétní, Measurable - měřitelný, Attainable - dosažitelný, Relevant - odpovídající, Time-bound - ohraničený v čase. Zdroj: http://cs.wikipedia.org/wiki/SMART_metoda

[5]pozn. autora: Lidové moudro



Obrázek 1: Aktivitní diagram zacházení s požadavky



Obrázek 2: Stavový diagram zacházení s požadavky

- Zrušen
- Možná/někdy[6]

3. U požadavku evidujeme historii

Aktivitní a stavový diagram zacházení s požadavky je zobrazen na obrázku 1 a 2.

2.1.2.1 Možnosti změn stavů požadavků V následujících odstavcích bude popsáno, jakým způsobem má být zacházeno s požadavky.

2.1.2.1.1 Upřesnění nejasného požadavku Jelikož požadavky sbíráme od obecnějších ke konkrétnějším, stává se, že první verze má k výsledné ještě daleko. Nicméně nezahazujeme historii, ale uchováme si vazbu starého požadavku na nový. Starý požadavek tedy změní svůj stav na "Nepřijatý, nejasný, upraven"a je vytvořen jeho následník, jenž by

[6]převzato z metodologie GTD jako stav věci, které bychom chtěli, ale v současné době nejsou reálné. Tyto požadavky jsou uloženy pro jejich dozrání nebo pozdějšímu zrušení požadavku.

přirozeně měl být jasnější. Tímto způsobem je potřeba požadavky vypilovat, abychom v pozdějších fázích projektu měli požadavky konkrétní, měřitelné, odsouhlasené[7].

2.1.2.1.2 Rozdělení požadavku Když se definují požadavky a od zákazníka přicházejí například emailem, mohou někdy být zmatené a musí být rozděleny na více požadavků. Vytváříme tedy odvozené požadavky na primární požadavek s tím, že si primární požadavek chceme uchovat v historii.

2.1.2.1.3 Nepřijatý, sloučen s jiným Inverzní operací k rozdělení požadavku je sloučení s jiným. Této operace můžeme využít, když se zdá, že by sloučení dvou požadavků mohlo být výhodné.

2.1.2.1.4 Zrušení požadavku Požadavky můžeme zamítnout. Do této skupiny patří nereálné požadavky, dále požadavky, které je možno řešit efektivněji, jinak. Zákazníkovi je potřeba taktně vysvětlit proč. Pokud je rozumný, takovéto řešení určitě přivítá. Pokud není, je na zváženou, zda opravdu chceme, s takovýmto člověkem spolupracovat.

2.1.2.1.5 Možná/někdy Tento stav byl převzat z metodologie GTD. Je to skupina stavů, jež v současné době nejsou důležité, a jsou uloženy pro možné pozdější zpracování, jakmile je zákazník bude vyžadovat.

2.1.2.1.6 Přijatý požadavek Poslední skupina požadavků jsou přijaté požadavky, které byly odsouhlaseny klientem i zhotovitelem (analytikem). Ty jsou určeny pro analýzu, implementaci, testování a nasazení.

2.1.3 Analýza aplikace

Dalším logickým krokem v rámci iterace je po specifikaci požadavků právě jejich analýza. V této fázi vytváříme diagramy modelující statické a dynamické aspekty aplikace, návrhy uživatelského rozhraní, aj.

Konkrétně tedy vytváříme:

1. diagramy (například v programu Violet)
    - diagramy případů užití (angl. Use case diagram)
    - třídní diagramy (angl. Class diagram)
    - objektové diagramy (angl. Object diagram)

[7] dle SMART kritéria

- stavové diagramy (angl. State diagram)
- aktivitní diagramy (angl. Activity diagram)
- sekvenční diagramy (angl. Sequence diagram)

2. případy užití
3. návrhy uživatelského rozhraní
    - tzv. drátěné modely (angl. wifeframe)
    - návrhy vzhledu formulářů a interakčních rozhraní

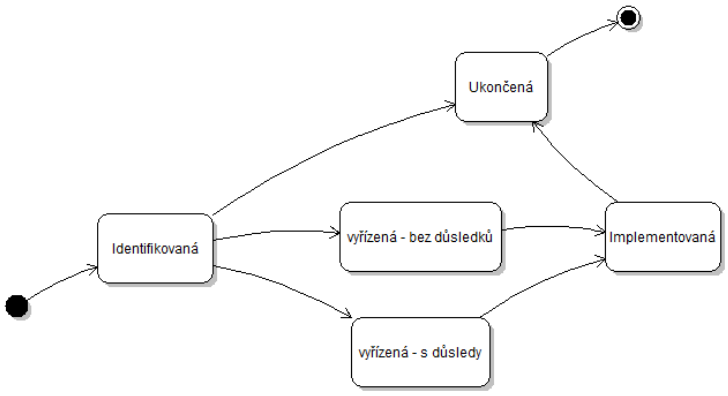
2.1.3.1 Propojení s aplikací Spec Veškeré tyto artefakty analýzy pak nahráváme do aplikace Spec. Ta přiřadí artefaktům jejich identifikátor a připojí je k požadavku na jejich základě artefakty vznikly. V rámci jednoduššího zpracování ve fázi implementace je vhodné nahrát diagramy ve formě obrázku. Při případných změnách požadavku, na jejichž základě vzniky artefakty analýzy, pak lze systematicky dohledat, kde se má provést změna v analýze a následně i v implementaci.

2.1.4 Řízení změn

Proces řízení změn je dle [4] definován jako činnost (metodika, proces), jejímž cílem je usnadnit a urychlit průběh změny tak, aby bylo možné efektivně realizovat zvolenou strategii a urychlit získání výnosů z vložených investic, kdy účinné dosažení změny předpokládá splnění více předpokladů a podmínek. K nejdůležitějším patří znalost metodiky provádění změn a manažerské schopnosti k účinnému řízení zamýšlené změny.

Definice procesu řízení změn podle ČSN ISO 10 007, [4] je na obecné úrovni:

1. Fáze: Identifikace změny
    - podnět
    - zpracování a předložení požadavku
    - analýza změny
    - schválení nebo neschválení změny
2. Fáze: Implementace
    - zavedení změny
    - monitorování změny
3. Fáze: Ukončení
    - Vyhodnocení



Obrázek 3: Stavy změn v aplikaci Spec

- Uzavření

2.1.4.1 Napojení řízení změn na aplikaci Spec Z důvodu potřeby řízení změn, byla navrhnuta změna, pro vyhledávání změn v aplikaci Spec. Tato změna byla pouze navrhnuta, nikoli implementována. Chybějící implementaci lze simuvat v jakémkoli tabulkovém procesoru. Změnu zde identifikuje jeden z požadavků z kapitoly 2.1.2.1 Možnosti změn stavů požadavků. Tato změna vytvoří v aplikaci novou entitu změna.

Pro tuto změnu pak můžeme evidovat následující informace:

- identifikátor změny
- datum a čas zanesení změny
- popis změn v ostatních souvisejících požadavcích
  Související požadavky jsou vysledovatelné v detailu požadavku. Je na uživateli-analytikovi, aby zanesl do systému, kterých požadavků se změna týká.
- stav změny

Samotná změna pak má stavy znázorněné v diagramu na obrázku 3.

2.1.4.2 Přínosy řízení změn Hlavní přínosy řízení změn je možnost dohledat, popsat a analyzovat změny. Samotné změny se pak dějí na úpravě požadavků a s nimi souvisejícími artefakty analýzy, které se dále projeví v samotné implementaci a testech v rámci nových iterací. Nově vzniklé požadavky a upravené požadavky mají status *nový požadavek*. Ti, jež zanáší změny rovnou do zdrojových kódů jsou odsouzeni ke stálému hledání souvislostí.

Obrázek 4: Narůstající složitost podnikání dle velikosti, převzato z [5]

2.2 Složitost zpracování obchodních příležitostí

Celou škálu možných obchodních příležitostí v IT bychom mohli rozdělit podle obrázku 4.

"Každý styl podnikání vyžaduje jiné know-how. Co funguje dobře na volné noze, nemusí platit při řízení středně velké firmy. A naopak. Každé podnikání si žádá svoje."[5]

Práve kvůli těmto faktům, byla navržena aplikace Spec maximálně minimalisticky, bez možnosti autentizace více uživatelů, časových výkazů práce či například sledování souvislostí tříd v třídních diagramech s ostatními diagramy, jako je tomu například v aplikaci Enterprise Architect.

2.3 Rizika zpracování obchodních příležitostí

O větší části rizik, kterých se můžeme dopustit, tedy technické části, se můžete dočíst v kapitole 6. V této kapitole bych uvedl rizika, jež je možné se dopustit, při samotném zpracování obchodní příležitosti. Tedy rizika vycházející ze samotného řízení projektu.

2.3.1 Časový dluh

Časovým dluhem bych označil situaci, která nastává uvolněním software nebo jeho části jež splňuje základní podmínky pro uvolnění za současného vypuštění některé z důležitých podpůrných procesů.

Jako podpůrné procesy jsou zde myšleny zejména tyto:

- byly provedeny změny fungování reálného programu oproti analýze a její nezaznačení do analýzy
- nedostatečná dokumentace
- ignorování chybových hlášení z místa, kde aplikace reálně běží
- nedostatečné a nepravidelné testování, na jehož základě se nám aplikace může vrátit jako vadná nebo nedostatečně funkční.

Chybu může udělat každý člověk. Nicméně profesionální přístup vyžaduje, aby takovýchto chyb bylo uděláno co nejméně a jakmile se chyba stane, musí se řešit.

Problém s časovým dluhem, je bohužel ten, že je potřeba jej řešit, až jakmile jsou vyčerpány zdroje na jeho řešení. Proto musí být do ceny a časového plánu, připočteny i náklady na občasné kontroly logů, testování a opravy chyb, jež se projeví.

2.3.2 Vyhrazení zdrojů pro vývoj diagnostických prvků provizního jádra

V rámci analýzy by měli být vyhrazeny zdroje pro návrh diagnostických prvků pro kontrolu finančních výpočtů aplikace. Implementací a zpřístupnění této funkčnosti provozovateli docílíme větší kontroly systému.

Jako příklad diagnostických prvků může sloužit:

- Kontrola přijmů a provizních výdajů z prodeje produktů. Výdaje obvykle nesmí být větší než přijmy

3 Optimalizace aplikace pro webový provoz

Důvodem optimalizací je snížení nároků na server, kde je aplikace nasazena.
Předpokladem této práce je návrh kompletně pro provoz na sdílených webových serverech. Z tohoto důvodu musí být aplikace navržena tak, aby vyhovovala následujícím omezením:

- omezení na doby běhu skriptu, typicky 30 sekund
- omezení na alokovanou paměť, typicky 64MB

Tato omezení se samozřejmě dají obejít využitím nesdílených hostingů, tj. virtuální serverů, dedikované serverů či cloudových řešení. Nicméně budeme se držet těchto omezení jako dnešního PHP standardu.

Aby nebylo těchto mezních hodnot dosaženo, optimalizujeme aplikaci podle postupů popsaných v následujících podkapitolách.

3.1 Optimalizace dotazů na databázový server

V této kapitole budeme řešit, možné návrhové chyby, kterých se můžeme při komunikaci s databázovým serverem dopustit. Tyto chyby mohou bránit provozu aplikace pod vyšším zatížením, takže chybu v návrhu nemusíme odhalit okamžitě.

3.1.1 Dotazy v cyklech

Dejme tomu, že chceme zjistit některé z informací z tabulky uživatelé. Budeme vyhledávat podle primárního klíče - id uživatele - pět uživatelů. Pokud tuto operaci bude provádět v cyklu, provede se pět dotazů na databázový server, který vrátí pět odpovědí. Například takto:

```
SELECT jmeno, prijmeni, ulice, mesto, psc FROM uzivatele WHERE id_uzivatel = 2;
SELECT jmeno, prijmeni, ulice, mesto, psc FROM uzivatele WHERE id_uzivatel = 7;
SELECT jmeno, prijmeni, ulice, mesto, psc FROM uzivatele WHERE id_uzivatel = 29;
SELECT jmeno, prijmeni, ulice, mesto, psc FROM uzivatele WHERE id_uzivatel = 44;
SELECT jmeno, prijmeni, ulice, mesto, psc FROM uzivatele WHERE id_uzivatel = 66;
```

Řešením tohoto problému je využití množinových operací, kdy klademe pouze jediný dotaz:

```
SELECT jmeno, prijmeni, ulice, mesto, psc FROM uzivatele
WHERE id_uzivatel IN (2,7,29,44,66);
```

V případě výčtu množiny o velkém počtu prvků nemusí být dotaz žádným způsobem dělen na části jelikož, empirickým testováním bylo zjištěno, že délka dotazu na databázi

PostgreSQL může mít velikost i 1MB, což stačí k výčtu 165 645 prvků s hodnotami jdoucími sekvenčně od čísla jedna. Reálná hranice délky dotazu nebyla předmětem hledání. Takovýto dotaz, který byl použit pro otestování, najdete v elektrinické příloze G.

Reálná situace, kdy bychom potřebovali jedním dotazem přenést, takto velké množství záznamů není v aplikaci potřebná, nicméně je znát limity navržených optimalizací.

3.1.2 Traverzování grafovou strukturou v programu

Tato optimalizace přesouvá traverzování grafovou strukturou na databázovou vrstvu, která je díky použitému jazyku C++ a interpretaci jazyka PL/PgSQL rychlejší než PHP.

Příklad 3.1
Mějme grafovou strukturu, kde je závislost potomek $\Longrightarrow$ rodič definována v tabulce uživatelé v atributech rodicovske_id $\Longrightarrow$ id_uzivatel. Traverzováním v aplikační vrstvě vznikají tyto dotazy a příslušné odpovědi:

```
SELECT id_uzivatel FROM uzivatele WHERE rodicovske_id = 67; /* vraci 66 */
SELECT id_uzivatel FROM uzivatele WHERE rodicovske_id = 66; /* vraci 44 */
SELECT id_uzivatel FROM uzivatele WHERE rodicovske_id = 44; /* vraci 29 */
SELECT id_uzivatel FROM uzivatele WHERE rodicovske_id = 29; /* vraci 7 */
SELECT id_uzivatel FROM uzivatele WHERE rodicovske_id = 7; /* vraci 2 */
```

■

Tuto situaci můžeme optimalizovat odsunutím traverzování z aplikační vrstvy do databázové vrstvy, které spočívá v napsání uložené procedury. Konkrétní řešení uložené procedury pro databázový server PostgreSQL 8.2+ najdete v příloze A.
S jejím využitím se komunikace mezi aplikačním a databázovým serverem omezí na:

```
SELECT * FROM getasc(67); /* vraci pole záznamů array(66, 44, 29, 7, 2) */
```

3.1.3 Hledání návrhových chyb

V hledání návrhových chyb může pomoci využití třídy *Nette\Diagnostics\Bar* z Nette Frameworku a využití databázové vrstvy dibi, jež zobecňuje komunikaci s hlavními databázovými servery. Třída *Nette\Diagnostics\Bar* ve vývojovém režimu zobrazí provedené dotazy, viz obrázek 5.

3.2 Ukládání informací do paměti cache

"Ukládání informací do paměti cache urychlí aplikaci tím, že jednou náročně získaná data uloží pro příští použití."[6]

Obrázek 5: Ladící panel z Nette Frameworku

Pro ukládání do tohoto typu dat, může být využita třída *\Nette\Caching\Cache* z Nette Frameworku, jež slouží pro ukládání dat (pole, instance tříd) v serializované podobně do souborů na webovém serveru.

Příklad 3.2
Vhodným příkladem použití *\Nette\Caching\Cache* je například uložení instance autentizační třídy. Typická autentizační třída obsahuje informace, kterým uživatelům podle role povolí nebo zakáže přístup k jednotlivým zdrojům podle uživatelských práv (zobrazení, zápis).

V jazyce PHP nelze udržet objekt v paměti serveru déle než je délka doby reakce na HTTP žádost. Proto je nutná inicializace autentizačního objektu s každým požadavkem na server. Z tohoto důvodu je vhodné využít mezipamět Cache pro uložení jednou vytvořeného objektu v serializované podobě a následně využívat tento objekt po jeho deserializaci.

Inicializace objektu tedy buď provede nutný počet dotazů do databáze, nebo načte zpracované data z *\Nette\Caching\Cache*. Samozřejmě v případě změn pravidel autentizace, je nutno pamět Cache invalidovat. ■

Příklad 3.3
V elektronické příloze F, naleznete jednoduchý příklad využití mezipaměti Cache. Jako složitěji konstruovaný objekt je zde pole čísel, u kterých lze ověřit prvočíselnost pomocí metody Wheel Factorization. Testujeme množinu čísel v rozsahu čísel od jedné do padesáti tisíc. Jestliže pomocí metody Wheel Factorization lze říci, že je číslo prvočíselné, je přidáno do pole.

Spuštění skriptu bez využití paměti Cache příkazem "*php run.php -nocache*"trvalo na počítači, kde byl skript vytvořen 2,75159 sekundy. Spuštění příkazem "*php run.php*"s využitím

paměti Cache trvala inicializace pole 0,00498 sekundy při velikosti 78,9kB serializovaných dat[8]. ■

[8]Předpokladem pro využití mezipaměti je, že data musí být vytvořena. Při smazání složky temp a spuštění s mezipamětí Cache se při prvním spuštění do paměti ukládá a při dalším spuštění z mezipaměti čerpá.

4 Automatické testování systému

Zejména u projektů většího rozsahu je vhodné z vývojového procesu nevynechávat fázi testování. Cílem testování, je *zajistit konstantní kvalitu cílového produktu.* Zásah do jednoho souboru může mít složité konsekvence i v jiných částech aplikace. Pokud na tyto konsekvence není kladen důraz, můžeme do produktu zanášet chyby.

Proč tedy testovat? Správným důvodem by mohlo být *udržení kvality produktu i po zásahu do jeho funkcionality*[9].

4.1 Současné možnosti testování

4.1.1 Použitelné technologie

V rámci technologií na jaké se tato práce zaměřuje, existujuje omezené množství testovacích nástrojů. V této práci bude popsán framework pro jednotkové testování PHPUnit. A dále techonologie Selenium pro simulaci interakce uživatele s prohlížečem, jež lze použít v kombinaci s PHPUnit. Jako další použitelné nástroje uvedu například IBM Rational Functional Tester, čímž by šel nahradit Selenium, bohužel je pro komeriční použití placený.

Dalším použitelným nástrojem je IBM Rational Performance Tester, kterým lze testovat zátěž na server. Bohužel se jedná také o komeriční produkt. Ten lze částečně nahradit pomocí *ab - Apache HTTP server benchmarking tool*, viz [9]. Výkonostnímu testování se v této práci nebudeme věnovat.

4.2 Typy možných automatizovaných testování

4.2.1 Regresní testování

"Účel metody regresního testování je zjistit, zda modifikace, provedené ve funkcionalitě, nezpůsobily tzv.regresi – rozbití zbývající funkcionality."[8] Regresní testování je zajištěno spouštěním testů po provedení modifikace ve funkcionalitě programu. Lze jej provádět ve všech testovacích úrovních a lze jej provádět automatizovaně. [8]

4.3 Testování výkonu a zátěžové testy

"Testování výkonu se provádí za účelem změření rychlosti, s jakou systém v určitém aspektu pracuje, při daném zatížení. Testování výkonu slouží k demonstraci faktu, že produkt splňuje výkonnostní kritéria. Zátěžové testování se používá k měření stability systému v

[9] a mít možnost večer ulehnout s klidným svědomím, že se nic důležitého nepokazilo

normálních i extrémních podmínkách, často až do bodu, kdy systém přestane fungovat. Cílem je pozorovat chování systému."[8]

4.4 Akceptační testy

Akceptační testy provádí zákazník. Nicméně simulace uživatelských scénářů lze simulovat a také automaticky testovat.

V rámci omezeného obsahu práce bych čtenáře majícího větší zájem o formy testování odkázal na Diplomovou práci Tomáše Kyjovského, viz reference [8].

4.5 Proč automatické testování?

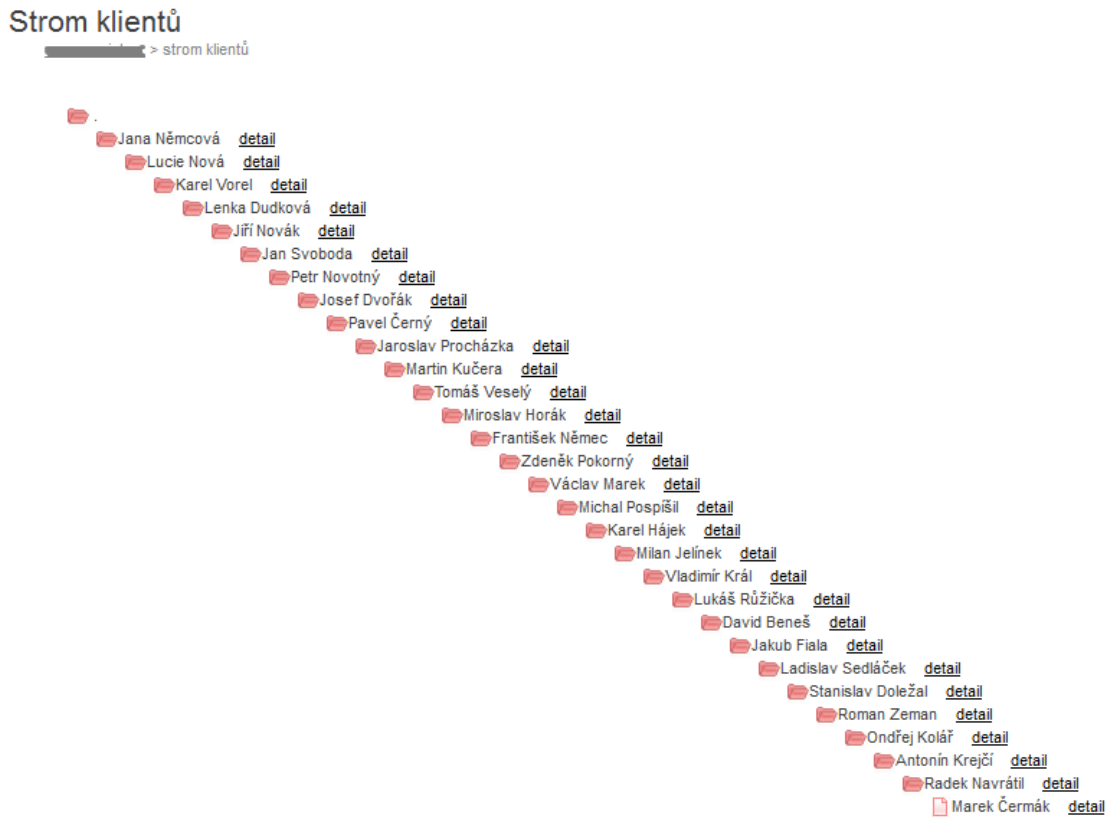
Testování přináší mnoho výhod, o které se nechceme ochudit. Manuální testování, je samozřejmě nejkvalitnější, jelikož dochází k interakci systému s uživatelem. Bohužel je časově náročné a navíc přináší vývojáři psychickou bolest při testování často opakovaných uživatelských scénářů. Zautomatizováním rutinních činností tedy zajistíme konstantní kvalitu systému efektivněji.

Testování jakožto fázi vývoje informačních systémů se zaměřením na MLM je jedna z klíčových, právě z důvodu, kdy je potřeba ověřit jak systém reaguje na uživatelské scénáře jako v příkladu 4.1.

Příklad 4.1
Potřebujeme ověřit výpočet provizí v modelové struktuře třiceti uživatelů. Každý uživatel je členem orientovaného grafu a vyjma posledního uživatele, má každý právě jednoho potomka. Takováto struktura je pro lepší představu zobrazena na obrázku 6.

Systém zaregistruje 30 uživatelů, aktivuje jejich konta a nakoupí každému stejný produkt. ■

```
<?php

namespace AntoninRykalsky\MoneyPoint;
use AntoninRykalsky as AR;

/**
 *
 * Generated by PHPUnit_SkeletonGenerator on 2013-01-02 at 13:31:05.
 * Testování modelů objednávek
 *
 * @author Bc. Antonín Rykalský <antonin.rykalsky@gmail.com>
 * @copyright  Copyright (c) 2009-2013 Antonín Rykalský
 * @link       http://mlm-soft.cz
 * @package    mlm-soft.cz
 */
```



Obrázek 6: Struktura třiceti klientů vhodná pro testování, příklad 4.1

```
class RegistrationThirty extends \PHPUnit_Framework_TestCase {

    var $levels = 30;

    /**
     * Pripravi strukturu
     */
    public function testMakeStructure() {

        AR\MoneyPoint\ApplicationSetup::get()->clear();
        AR\MoneyPoint\ApplicationSetup::get()->setupProducts();

        $users=array();
        for( $i=0; $i<$this->levels; $i++ )
            $users[$i] = ApplicationPersonGenerator::get()->getPerson( $i );

        AR\MoneyPoint\ApplicationSetup::get()->setupUsers( $users );
    }

    /**
     * Nakoupí produkty
     */
    public function testBuyCreditsForMembers()
    {
        $quantity = AR\MoneyPoint\ApplicationSetup::get()->creditsDistribution[count(AR\
            MoneyPoint\ApplicationSetup::get()->creditsDistribution)-1]['credits'];

        $goods = new \MoneyPoint\OrderGoods();
        $goods->addToBasket(1, $quantity);

        $details = new \MoneyPoint\OrderDetails();
        $details->setPayment(3);

        for( $idu=1; $idu<=$this->levels; $idu++ )
        {
            $member4 = new \MoneyPoint\Member( $idu );
            $idOrder1 = $member4->credits()->buy( $goods, $details );
            $this->assertTrue(is_numeric($idOrder1));
            $order1 = new \MoneyPoint\CreditOrder( $idOrder1 );
            $this->assertTrue( $order1->order->id_status === 1);
            $this->assertTrue( $order1->order->order_timestamp instanceof \DibiDateTime );
            $order1->setPaid();
            $this->assertTrue( $order1->order->id_status === 3);
            $order1->setRewarded();
        }
    }


}
?>
```

Výpis 1: Testovací skript z příkladu 4.1

Příklad 4.1, je typově ideální pro ověření funkce základních početních faktů. Modelová situace by měla být odsouhlasena zadavatelem. Na tomto příkladu můžeme ověřit výši provize pro každého uživatele, vidíme dobu zpracování webovým serverem.

Tento scénář můžeme nahradit testovacím skriptem 1, který byl využit pro vývoj řízený testy. Ověření výše provizí již bylo řešeno manuálně, podle ručně vypočtených scénářů.

Příklad 4.2
V dalších iteracích softwarového procesu pak rozšiřujeme původní příklad. Nákupy z př.4.1 rozdělíme do tří účetní období. Každé s vlastní uzávěrkou. Zde ověřujeme přenášení nevyplacených provizí, které vzniknou nedosažením minimální výše k vyplacení. Tento příklad je pouze ilustrativní, není součástí práce. ■

Zde už provedení testu manuálně nepřichází v úvahu, protože by práce byla rozprostřena do velkého časového období.

4.5.1 Testy nasazení

Testování nasazení aplikace na ostrém serveru, je vhodné zejména z důvodu odlišného prostředí, než na jakém byla aplikace vyvíjena. Toto prostředí se může lišit v nastavením direktiv a modulů aplikace Apache či PHP. Server může mít jinak nastaveny direktivy Apache či PHP a výsledný produkt se může chovat jinak než na lokálním/vývojovém serveru.

Testy nasazení můžeme provádět pomocí nástroje Selenium z lokálního serveru a naprogramovat rozhraní, jež bude zprostředkovávat přístup k databázi, kde lze ověřit provedení operací. Bohužel tímto do aplikace zanášíme potencionálně zranitelné místo, proto je potřeba přenos šifrovat[10].

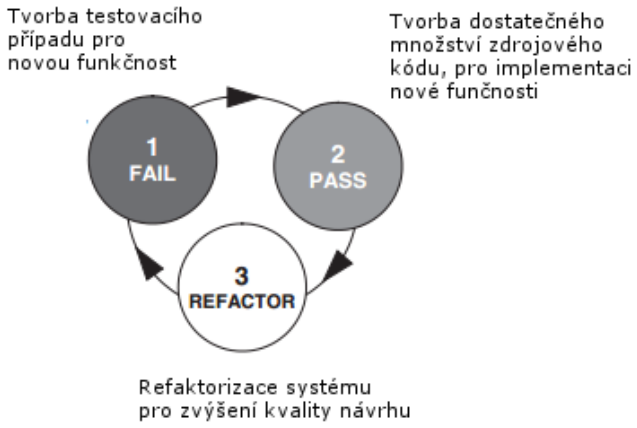
Testování nasazení není vhodné provádět na produkčním serveru, kde funguje ostrá verze aplikace. Můžeme ale zprovoznit identickou kopii ostré aplikace na skryté subdoméně, například na subdoméně *beta*.

4.6 Procesy podporující testování

4.6.1 Vývoj řízený testy

Testy řízený vývoj neboli TDD je přístup k vývoji software, jež probíhá v krátkých iteracích založených na automatickém regresním testování. TDD byl popularizován metodikou extrémního programování, která z něj udělala stěžejní přístup. [7]

[10] například pomocí knihovy mcrypt z php



Obrázek 7: Schéma průběru prací při použití TDD
Zdroj: [7] st.3, vlastní úprava

Při jeho použití předchází tvorba testů samotnou produkci kódu. Schéma průběhu prací se řídí dle schématu na obrázku 7. [7]. Jakmile je tedy zdrojový kód upraven do takového stavu, aby byly testy úspěšně vykonány, máme dle této metodiky provést refaktorizaci systému. Metodika TDD tedy do vývoje zanáší příznivé návyky.

TDD je často mylně chápána jako technika pro zajištění testování kvality, ačkoli se jedná o způsob vývoje software. Testování je pro TDD spíše prostředkem než účelem. [7] str.5.

V této práci je TDD zařazeno jako *proces podporující testování* z důvodu, že vedlejším produktem tohoto úsilí je množství testovacích tříd, jež jsou schopné odhalit nedostatky související se zásahem do zdrojových kódů aplikace a tím nepřímo podpořit udržení konstantní kvality aplikace.

4.6.2 Průběžná integrace

Okolo vývoje softwaru existují spousty doporučení, jak jej dělat správně nebo lépe. Jedním z takovýchto doporučení je pravidelný a automatický proces sestavení aplikace (angl. build) a proces otestování sestavené aplikace. Pro tuto praktiku použijme termín *Průběžná integrace* (angl. Continuous Integration, zkr. CI), jež je jeden ze stěženích částí extrémního programování. Tuto techniku lze využít samostatně bez zapojení ostatních částí extrémního programování.[10]

"Průběžná integrace je proces vývoje software, kde členové týmů integrují jejich práci do projektu pravidelně, každý člen často integruje alespoň denně nebo častěji. Každá integrace

je ověřena automatickým sestavením, které zahrnute testování k nalezení integračních chyb, jak jen nejrychleji je to možné."[11][11]

Z kontextu kapitoly *Procesy podporující testování*, ve které se nacházíme, se zde zaměříme právě na detekci chyb.

CI definuje, že by projekt měl obsahovat samotestovací kód, tzv. verifikační testy sestavení (angl. Build Verification Tests, zkr. BVT). BVT jsou sestavovány samotnými programátory, k ověření zda jejich kód funguje či ne. Mějte prosím na paměti, že BVT nenahrazuje klasické testování, z pohledu QA. BVT by měly být provedeny dříve, než se aplikace dostane k ověření do oddělení QA. Nicméně součástí BVT se můžou stát i akceptační testy a testy z pohledu QA. [10]

Obecné schéma prácí s použitím průběžné integrace je uvedeno na obrázku 8.

4.7 Používané softwarové nástroje

4.7.1 Selenium IDE

Selenium je testovací nástroj, určený k testování webových aplikací. Selenium umožňuje provádět série příkazů, které simulují aktivitu uživatele v interakci s webovým prohlížečem. Technologie Selenium je postavená nad jazykem javascript a zpracováním DOM struktury dokumentu.

Tento nástroj je vhodný k ověření průchodů uživatelskými scénáři. Lze tak simulovat akceptační testy zákazníkem. Je vhodný k automatizovanému ověření funkčnosti komunikace mezi softwarovými vrstvami Controller a Model. Pro testování business logiky v rámci TDD se nehodí z důvodu pomalého spuštění. Tehdy je lepší využít klasický pouze PHPUnit.

4.7.2 PHPUnit

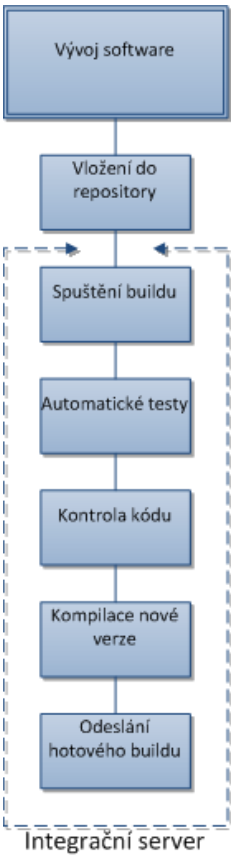
PHPUnit je framework pro jednotkové testování aplikací v PHP. Jednotkové testování je činnost, při které automatizovaně ověřujeme, že se aplikace chová tak jak předpokládáme. [14] str.1

K tomu slouží sada metod assert_*, [12] pomocí kterých ověřujeme výstupy funkcí, kterým jsou předány simulační vstupy.

Spouštění PHPUnit testů je relativně rychlé, hodí se pro TDD, kdy je potřeba testy spouštět pravidelně.

[11] Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.

[12] assertTrue, assertEquals, aj. viz manuál [14]



Obrázek 8: Schéma práce s integračním serverem, *Zdroj: http://cs.wikipedia.org/wiki/Soubor:Prubezna-integrace.png*

4.7.3 Jenkins CI

"Patrně nejznámějším a nejvíc používaným[13] CI serverem je Jenkins. Server je napsaný v Javě a původně byl určen rovněž pro kontinuální integraci projektů napsaných v Javě. Pochopitelně tedy obsahuje velmi silnou podporu nástrojů určených pro projekty napsané v Javě, nicméně je možné jej bez problémů použít i pro projekty v jiných jazycích. S růstem popularity (jak kontinuální integrace, tak Jenkins CI) roste i počet pluginů, které podporují nástroje určené pro jiné jazyky než Java. Toto je ostatně jedna z velkých výhod Jenkins CI oproti jiným CI serverům - komunita vývojářů (projekt je pochopitelně open source) je veliká a velmi aktivní (projekt má v tuto chvíli na githubu[14] 112 vývojářů a obsahuje 555 repozitářů), v nabídce několika set pluginů lze najít plugin pro kde co a stejně tak na mailing listu obvykle dostanete odpověď velmi rychle. Projekt je primárně určen pro malá nasazení (několik projektů a několik vývojářů), nicméně lze jej s úspěchem použít i pro obrovské nasazení (stovky/tisíce projektů, jedno z větších veřejných nasazení je např. build server ASF)."[13]

[13]Podle průzkumu společnosti Sonatype 83% respondentů používá kontinuální integraci a z nich 71,5% používá (tehdy ještě) Hudson.

[14]poznámka autora: úložišti technologie GIT na https://github.com/

5 Modulárnost aplikace

Jedním z požadavků, které jsou kladeny na proces výroby těchto aplikací je zajištění modulárnosti aplikace. To znamená, že cílový systém musí být složen z nutného počtu logických prvků[15], které zajistí potřebnou funkci systému. Pro informaci, dle [15], opačným stavem by byla molochální aplikace, jež se obtížně testuje a její funkcionalita není nijak jasně ohraničena.

Co je to tedy modul? Pojem modul je možno chápat v různých kontextech jinak. Může se jednat o komponentu, pracující nad jednou tabulkou, či například modul administrace jež má v konečném důsledku obsluhovat celý systém. V této práci chápejme pojem modul v jeho abstraktním slova smyslu, jako blok, který vykonává svou jasně danou omezenou funkci.

Jako klasické přirovnání, lze použít automobilový průmysl. Automobil Škoda Superb, je vyráběn na podvozku, který je použit i pro automobily Volkswagen Passat. Superb je dodáván ve čtyřech různě výkonných variantách benzínových motorů a čtyřech variantách dieselových motorů[16].

5.1 Návrh a vývoj modulu

Modul musí splňovat tyto podmínky:

1. Musí být oddělitelný od aplikace, nenásilným způsobem.
2. Modul musí mít jasně definované závislosti

Samotný návrh modulů, by měl být realizován ve fázi analýzy systému. Dekompozicí systému na menší funkční celky lze vhodně ilustrovat na modelu třídního diagramu s využitím prvků *package*. Existují i alternativní způsoby, jež souvisí například se strukturálním a funkcionálním programováním.

Při implementaci pak využíváme prvky jazyka, v tomto případě PHP 5.3. Ten umožňuje používat všechny dnes typické rysy OOP, jako jsou jmenné prostory, dědičnost, zapouzdření aj. Od verze PHP 5.4 lze využívat i tzv. *traits*, kterým lze simulovat vícenásobnou[17] dědičnost. V rámci implementace je, je důležité adresářově oddělit jednotlivé moduly. Způsoby jaké k tomu můžeme využít popisují kapitoly *Nette MVC moduly*, viz 5.2.1 a *Správa závislostí pomocí nástroje Composer*, viz 5.3.1

[15] Logickými prvky je myšleno modulů či pluginů chcete li.
[16] anon.
[17] dědění potomka od více předků

5.2 Architektura aplikace

Architektura aplikace musí modularitu nutně podporovat. Jak již bylo několikrát zmíněno pro implementaci byl využit Nette Framework, který oproti použití čistého PHP přináší řadu vlastností, na které jsme zvyklý z jiných-modernějších technologií.

Jedním z nich je například RobotLoader, který načítá všechny soubory php v definovaných adresářích a tím odstraňuje nutnost fixně definovat cesty mezi jednotlivými soubory, jak se často dělo pomocí PHP příkazu *include*[18]. Dalším přínosem je implementace architetury MVC.

5.2.1 MVC architektura

"Model-View-Controller je softwarová architektura, která vznikla z potřeby oddělit u aplikací s grafickým rozhraním kód obsluhy (controller) od kódu aplikační logiky (model) a od kódu zobrazujícího data (view). Tím jednak aplikaci zpřehledňuje, usnadňuje budoucí vývoj a umožňuje testování jednotlivých části zvlášť."[16]

5.2.1.1 Nette presentery Základní vrstu aplikace tvoří business logika. Tu je potřeba napojit na interaktivní uživatelské rozhraní. V rámci této práce je rozhraní pro interakci tvořeno formou HTML stránek, které poskytují uživatelsky přívětivější rozhraní. Nette framework se stará o mapování uživatelských akcí na presentery a akce kde je požadavek obsluhován.

Každý požadavek na naši aplikaci se dostane přes soubory index.php a bootstrap.php do objektu \Nette\Application $application. Objekt $application je předá k přeložení službě *router*. V případě, že presenter existuje $application požádá službu *presenterFactory* o vytvoření presenteru, jež požadavek obslouží."[16]

K celým tímto odstavcem míříme k tvorbě modulů, tedy jak říká dokumentace Nette, u složitějších aplikací můžeme složky s presentery a šablonami rozčlenit do podadresářů, které Nette nazývá moduly. Moduly nemusí tvořit jen plochou strukturu, lze vytvářet i submoduly atd. [16]

Následuje příklad struktury Nette MVC aplikace:

```
multilevel-projekt-abc/
    app/                ? adresář s aplikací
        models/         ? modelová vrstva a její třídy
        AdminModule/    ? modul Admin
            ClientsModule/     ? modul Clients pro správu klientů
             presenters/  ? jeho presentery
             templates/   ? a šablony
```

[18] a jeho alternativ include_once, require, require_once.

```
        CommissionModule/    ? modul Commission pro práci s provizemi
         presenters/  ? jeho presentery
         templates/   ? a šablony

    FrontModule/    ? modul Front pro veřejnou část webu
        presenters/  ? jeho presentery
        templates/   ? a šablony

    bootstrap.php  ? zaváděcí soubor aplikace
...
```

Tyto MVC moduly jsou adresářově oddělené a lze je nenásilným způsobem od zbytku aplikace oddělit a v případě potřeby spravovat mimo hlavní projekt.

5.3 Správa závislostí

Existují různé způsoby jak evidovat vyvíjené projekty. Jako předchůdce dnešních nástojů lze uvést například CVS nebo Subversion. V nástroji Subversion například jde definovat pomocí vlastnosti *svn:external* vazby z jiných projektů. Případně lze v procesu integrace pomocí operací SVN Merge, slučovat subprojekty do sebe, což lze řešit automaticky v procesu integrace například pomocí ant skriptů.

V této sekci se budeme věnovat modernímu ekvivalentu těchto nástrojů.

5.3.1 Správa závislostí pomocí nástroje Composer

5.3.1.1 Co je to Composer Composer je nástroj pro správu závislostí nad jazykem PHP. Umožňuje deklarovat závislosti na knihovnách, které projekt vyžaduje a umožňuje je do projektu automatizovaně nainstalovat. Composer je silně inspirován projekty NPM[19] a Bundler Ruby[20]. [18]

5.3.1.2 Použití nástroje Composer Pro použití nástroje Composer musíme nadefinovat hlavní projekt a jeho závislosti na ostatních knihovnách. Každá knihovna by měla být uložená v repositáři Git (případně jiném ver, balíku zip, aj.) spolu s konfiguračním souborem *composer.json*, který identifikuje:

- název balíku
- popis balíku
- verzi

[19] poznámka autora: Node Packaged Modules, https://npmjs.org/

[20] poznámka autora: balíčkový systém Bundler používaný ve frameworku Ruby pro jazyk Python

V hlavním projektu pak máme uložen soubor *composer.json*, který opět identifikuje balíček stejným způsobem a dále definuje závislosti na ostatních balíčcích, případně konkrétních verzích.

Příklady konfiguračních souborů uvádím v elektronické příloze C. Příklad závislostí balíku na hlavním projektu uvádím v příloze B.

5.3.1.3 Automatizace Po nadefinování závislostí lze využít příkazy *composer install* pro instalaci nejnovějších balíčků do hlavního projektu a *composer update* pro aktualizaci balíčků na jejich nejnovější verzi.

Composer tímto vloží nebo aktualizuje, moduly do předem stanovené složky (například do libs nebo standardně vendor). Umístění těchto skriptů mimo hlavní aplikaci (složku app), není problém protože můžeme využít buď standardního autoloadingu PSR-0 nebo RobotLoader frameworku Nette.

Hlavním přínosem tedy je možnost oddělit funkční část modelů, která může být znovupoužitelná a její odsun do externího repositáře. Příklad v elektronické příloze C definuje použití knihovny *AntoninRykalsky/ApplicationBase*. Jak již bylo řečeno, hlavním přínosem je jednoduché a plně automatizované vložení do různých projektů.

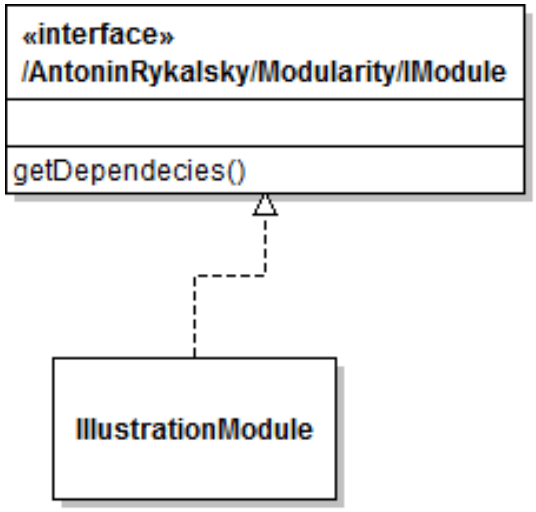
5.3.2 Zacházení se závislostmi na externích souborech

Jelikož se pohybujeme v kontextu webových aplikací, je obvyklé, že potřebujeme vytvořit modul více komplexnější, jako například správa uživatelských rolí. Tento modul by potencionálně mohl být využit ve více projektech. Pokud se rozhodneme do modulu zařadit i uživatelské rozhraní, může se stát, že budeme potřebovat do modulu vložit kromě zdrojových kódů PHP, například i client-side javascript kódy, obrázky, kaskádové styly, flash objekty či databázovou strukturu a data v databázi.

Databázovou strukturou se zabývá kapitola *Správa databázových závislostí*, viz str.5.4.

V tomto odstavci bude popsán návrh, jak tyto závislosti řešit. Soubory (javascript soubory, obrázky aj.) by měli být součástí modulů. Problémem je, že složky libs nesmějí být přístupné z webového rozhraní, což je řešeno direktivou *Deny from all* v *.htaccess* pro server *Apache* nebo alternativou pro server *IIS*. Právě tyto soubory, ale musí být přístupné z webového rozhraní. Řešením je tedy rozkopírovat potřebné soubory do adresářů, jež jsou definovány v hlavním projektu, podle typu například:

- $\%js\_dir\%$ jako /js
- $\%images\_dir\%$ jako /images
- $\%styles\_dir\%$ jako /styles
- $\%files\_dir\%$ jako /files



Obrázek 9: Společné rozhraní pro třídy, popisující umístění závislých souborů

Další částí řešení je automatizace rozmístění souborů na požadované umístění. To může být provedeno umístěním třídy, která popisuje zdrojové a cílové umístění souborů, které daný modul využívá, do modulu. Obrázek 9 ilustruje, způsob jakým mají být závislosti definovány. Následně algoritmem v PHP skriptu spuštěným z konzole obsahujícím následující kód pro zjištění, které třídy z rozhraní *\AntoninRykalsky\Modularity\IModule* dědí, můžeme dané soubory rozmístit na požadované umístění. Tento algoritmus bude obsahovat následující kód:

```
$classes = get_declared_classes();
$implementsIModule = array();
foreach($classes as $klass) {
   $reflect = new ReflectionClass($klass);
   if($reflect->implementsInterface('IModule'))
      $implementsIModule[] = $klass;
}
// prevzato z http://stackoverflow.com/questions/3993759/
php-how-to-get-a-list-of-classes-that-implement-certain-interface
```

Následně skript bude obsahovat logiku pro samotné rozmístění souborů.

5.4 Správa databázových závislostí

Tato podkapitola se zabývá otázkou jakým způsobem řešit těsnou vazbu mezi strukturou a daty v databázi s zdrojovými kódy v rámci modulů, které chceme znovupoužívat.

V úvodu kapitoly, bylo uvedeno, že by modul měl obsahovat i databázovou strukturu a data v databázi. Samozřejmě pouze v případě, že je na těchto datech modul závislý. Na

moduly v této kapitole je potřeba nahlížet z jiného hlediska, než tomu bylo v dřívejších podkapitolách. Zde jde o logické celky, které chceme využívat ve více aplikacích.

Myšlenka správy databázových závislostí z pohledu modularity je postavena na vykonávání dotazů v časové posloupnosti, jak dotazy vznikaly a zároveň podle toho, které moduly byly pro instalaci vybrány. Pro lepší představu, je zobrazeno uživatelské rozhraní pro instalaci databázových modulů na obrázku 10. Tyto dotazy primárně vytvářejí strukturu databázových tabulek, klíče, indexy a vkládájí výchozí hodnoty dat. Komplexní funkčnost si můžeme představit jako konečný automat, kdy počáteční stav je prázdná databáze a postupným prováděním zvolených dotazů databáze mění svůj stav, čímž se přibližuje k chtěnému stavu.

Každý dotaz je zde identifikován číslem modulu, verzí modulu a časovým razítkem ve formě *unix time stamp*[21]. V rámci instalace databázových závislostí do projektu evidujeme, které moduly byli nainstalovány a v jaké verzi. V případě, že v rámci modulu aktualizujeme některá data, vytvořením nového dotazu, můžeme data distribuovat i do ostatních projektů a provést pouze změny, které ještě nebyly provedeny.

V rámci automatizace rutinních činností při znovupoužívání takovýchto modulů (logických celků) vznikla aplikace *setup* jež je součástí elektronické přílohy F.

Využití
Využití je výhodné při dvou situacích:

- instalaci/aktualizaci databázové struktury při použití znovupoužitých modulů do nového projektu
- změny databáze při nahrávání změn na ostrý server

Výhodné je také, užití při nahrávání změn v databázi na ostrý server, kdy si můžeme celý projekt spustit na subdoméně ve stavu jako je na ostrém serveru. Dále doinstalujeme změny a otestujeme aplikaci. V případě, že aplikace běží podle očekávání, provedeme identické operace na ostrém serveru.

[21] počet vteřin od 1.1.1970

Instalátor modulů [beta.moneypoint.cz]home

K nainstalování

Zobraz detaily modulů

id	název modulu	instalovaná verze	nejnovější verze	akce	ignoruj
1	Konfigurace	17	13	OK	☐
2	ACL	2	1	OK	☐
3	CMS	5	5	OK	☐
6	Base cleaner	1	1	OK	☐
7	Uživatelé	3	3	OK	☐
8	Terms	1	1	OK	☐
9	Eshop base	11	11	OK	☐
10	Uživatelé: pass-recovery, waiting, abadon	4	4	OK	☐
13	Státy, obce	1	1	OK	☐
18	Uživatelé members - emaily při registraci	2	2	OK	☐
21	MLM Traverzování - fixní	2	2	OK	☐
22	Menu klient v adminovi	1	1	OK	☐
23	Kategorie - přirážka k prodejní ceně	1	1	OK	☐
91	Moneypoint	2	2	OK	☐

Ignorované

id	název modulu	instalovaná verze	nejnovější verze	akce	ignoruj
4	Báňská liga	-	3	upgrade	☑
5	Galerie	-	-	OK	☑
11	Bankovní konta	-	2	upgrade	☑
12	Bodová konta	-	1	upgrade	☑
14	Účetní období	-	1	upgrade	☑
15	ArcoAurum namíru	-	2	upgrade	☑
16	ArcoAurum namíru - číselník státy	-	1	upgrade	☑
17	ArcoAurum namíru - obce, kompletní	-	1	upgrade	☑
19	ArcoAurum namíru - testovací data	-	2	upgrade	☑
20	ArcoAurum namíru - klientské menu	-	1	upgrade	☑
24	Mlmsoft - nastaveni	-	1	upgrade	☑
90	Radosti zivota	-	12	upgrade	☑

Změň ignores ! | Instaluj moduly !

Fix sequences !

Obrázek 10: Uživatelské rozhraní pro instalaci databázových modulů

6 Eliminace výskytu bezpečnostních chyb a jejich zneužití

V této kapitole se budeme věnovat bezpečnostní stránce informačních systémů, jemiž se tato práce zabývá. V první řadě seskupíme možné potencionální hrozby do skupin se společnými vlastnostmi. Poté budou jednotlivé hrozby popsány, budou popsány rizika a opatření jak zabránit zneužití zranitelných míst.

6.1 Identifikace bezpečnostních rizik

6.1.1 Rizika spojená s využitím neznalosti vývojáře

Tuto skupinu chyb lze často najít v rukopisu začínajících vývojářů. V dnešní době a s využitím pokročilejších technologií nejsou už tak aktuální, což je důvodem, že se jimi práce nebude zabývat. Nicméně pokud systém, stojí na špatných základech, nejedná se o dobrý systém.

6.1.1.1 SQL injection SQL injection je napadení vedené předáním takových parametrů aplikaci, aby dotaz na relační databázi zprostředkoval útočníkovi chtěná data, odstranil nějaké data z databáze či data v databázi upravil. Základní informace co je to SQL injection a jak jej ošetřit, lze najít v dokumentaci PHP, viz [19].

Při využití databázové vrstvy *dibi*, nelze standardně provést více SQL příkazů v jednom dotazu, což je obrana proti vstupům typu:

```
Robert'); DROP TABLE Students;--
```

Navíc v dibi při skládání dotazů je vhodno, nevyužívat skládání řetězců, ale standardních funkcí, které skládají dotazy podle modifikátorů string, integer, aj.:

```
dibi::query('UPDATE `table` SET ', $arr, 'WHERE `id`=%i', $x);
// převzato z http://dibiphp.com/cs/quick-start
```

6.1.1.2 Cross-Site Scripting "Cross-Site Scripting je metoda narušení webových stránek zneužívající neošetřených výstupů. Útočník pak dokáže do stránky podstrčit svůj vlastní kód a tím může stránku pozměnit nebo dokonce získat citlivé údaje o návštěvnících. Proti XSS se lze bránit jen důsledným a korektním ošetřením všech řetězců."[20]

Možné hrozby:
Pomocí XSS hrozí infiltrace na konta uživatelů systému. Vytvořením skriptu, který pomocí příkazu *document.location* jazyka javascript přesměruje na stránku vytvořenou útočníkem, která bude vzhledově identická s přihlašovacím formulářem do klientské zóny, může útočník získat přihlašovací údaje klienta.

funkce	popis
htmlspecialchars	převede speciální znaky na HTML entity
json_encode	převede speciální znaky pro JSON/javascript reprezentaci

Tabulka 2: Funkce jazyka PHP pro escape proměnných

Opatření:
"Obranou je důsledné escapování vypisovaných dat, tj. převod znaků majících v daném kontextu speciální význam na jiné odpovídající sekvence."[21]

Základní opatřením je tedy escapování vstupních dat. V jazyce PHP pro to máme sadu funkcí z tabulky 2.

V Nette Frameworku je zabudována technologie *Context-Aware Escaping*, která převádí proměnné tímto způsobem automaticky transformuje vypisované proměnné v závislosti na kontextu. Tj. mezi tady *<script></script>* transformuje proměnné pomocí funkce *json_encode* a v klasickém HTML pomocí funkce *htmlspecialchars*. Je zde použita opačná logika než nabízí čisté PHP, tj. co není označeno, že se nemá transformovat, transformováno bude.

Příkladem potencionálně nebezpečného vstupu je následující:

```
<script type="text/javascript">
document.location = 'http://www.fake.com';
</script>
```

Očekávané chování je, aby HTML prohlížeč zobrazil výstup v identické podobě se vstupní podobou. Tj. vstupní data musí být transformovány do této podoby:

```
&lt;script type=&quot;text/javascript&quot;&gt;
document.location = 'http://www.fake.com';
&lt;/script&gt;
```

6.1.1.3 Další časté chyby

Mezi další časté chyby se řadí neošetření aplikace například na *Cross-Site Request Forgery*, *URL attack*, *control codes*, *invalid UTF-8*, *Session hijacking*, *session stealing* či *session fixation*. Těmto chybám zabranuje použití funkcí z Nette Frameworku. Příklady a popisy lze najít v dokumentaci, viz reference [20].

6.1.2 Zranitelnost autentizačních formulářů

Přes autentizační formuláře, může být veden útok na konta uživatelů, jež mají slabší složitost hesel. Tyto útoky, lze provádět pomocí slovníků nejčastějších hesel či bruteforce útoků. Zde je částečnou obranou, odmítnutí nebo komplikace přihlášení při určitém počtu špatných pokusů.

Jako možnosti ke komplikaci přihlášení můžeme využít captcha technologie, tj.opisování alfanumerického kódu z obrázku[22] nebo časové zabránění dalšímu přihlašovnání. Možnosti odmítnutí přihlášení je blacklisting podle IP adresy.

6.1.3 Ochrana dat patřících autentizovaným uživatelům

V rámci návrhu aplikace je důležité, aby klientská data, mohla být zobrazena pouze klientovi, kterému patří. Místem kde toto může být snadno opomenuto, může být uvedeno například stahování generovaných dokumentů (faktury, aj.) ze souborového serveru, kdy změnou ID artefaktu v URL požadavku, může dojít k pokusu o stažení dokumentu patřícího jinému klientovi.

6.1.4 Útoky na přístup k databázi

Dalším potenciálně zranitelným místem k databázi je její administrace. Její URL adresa bývá u veřejných hostingů známá a veřejně přístupná. Zde hrozí možný brute force útok, který nemůžeme ovlivnit. Je tedy důležité zvolit takové heslo, jehož prolomení bude náročné pro brute force generátory a slovníkové útoky.

Zde nabízí řešení aplikace *KeePassX*, která slouží pro správu hesel. Součástí této aplikace je generátor náhodných kódů, jež dokáže pomocí entropického generačního procesu vygenerovat náhodné heslo složené z malých a velkých písmen, čísel a speciálních znaků jakékoli délky. Pomocí této aplikace můžeme heslo uložit a následně kopírovat pro ulehčení procesu zadávání hesla. V rámci uložení hesla lze nastavit i upozornění, že heslo používáme již nějakou dobu a je načase jej změnit a tím prolomení ještě více zkomplikovat.

6.1.5 Rizika spojená s interními útoky na aplikaci

Tato rizika jsou spojená s možným interním útokem ze strany zaměstnance webhostingu. Osoba, která má na starosti webový server, musí mít plný přístup i na server kde jsou uloženy zdrojové kódy aplikace, i všechny generované artefakty (faktury, objednávky, aj.), tak i do databáze.

Možné hrozby:

1. Zvýšení provizního konta uživatele, jež je klientem nebo v kontaktu s klientem společnosti provozující daný projekt.
2. Dočasné změny čísla bankovního účtu v momentě kdy dá očekávat úhrada klienta, tj. před generováním obrazovky rekapitulace objednávky.

[22] Obrázek by měl být generován, takovým způsobem, aby se zabránilo jeho čitelnosti technologiemi optického rozpoznávání znaků - OCR. Jako prvek captcha, by nemělo být například sečtení dvou čísel jako je tomu u *http://posta.vsb.cz/horde*. V rámci komfortnosti používání by opsání kódu mělo být vyžadováno až po několika nezdařených pokusech.

Obrázek 11: Schéma šifrování citlivých dat a vrstvy aplikace

3. Jiné zneužití dat klientů

Detekce zneužití hrozby:
V návrhu aplikace by mělo být počítáno s hospodářskou finanční kalkulací. Klientská společnost finanční prostředky nevytváří, pouze je kumuluje a zprostředkovává dále. Vedení klientské společnosti by si mělo přát vidět finanční rozvahu a v případě nejasností samo provádět kontrolu, komu proč finanční rozvaha nesedí.

Základní opatření:
Předpokladem je, že tento typ hrozby zanechá stopu, kterou je možno dosledovat až k osobě, která si finančně přilepšila. Základní opatření předpokládá řešení pomocí občanskoprávních vztahů zaměstnance a webhosterské společnosti a smluvních vztahů webhosterské společnosti a jejího klienta.

Pokročilejší opatření:
Veškerá data lze šifrovat. V programovacím jazyce PHP k tomu lze využít například knihovnu *Mcrypt*[23], nebo *openSSL*. Data v databázi či artefakty na souborovém serveru pak lze ukládat v šifrované podobě a zpřístupnit ji pouze autorizovaným uživatelům.

Problém nastává v momentě, kdy v rámci návrhu aplikace chceme uložit šifrovací/dešifrovací klíč nebo vlastní šifrovací/dešifrovací funkci. Schéma je zobrazeno na obrázku 11. Jelikož zde zdrojové kódy aplikace nejsou ani kompilovány, jsou bohužel pro možného útočníka velice transparentní. Zde přichází s řešením produkt *ionCube*[24], jež kompiluje a šifruje zdrojové kódy, které již pak nelze zpětně dešifrovat do původní podoby. Lze je pouze spustit na serveru s nainstalovaným rozšířením *ioncube_loader* v *PHP*. Tímto způsobem pak musí být zašifrovány minimálně všechny zdrojové kódy, které pracují s šifrovanými daty. Dle obrázku tedy šifrujeme aplikační vrstvu.

Proces šifrování zdrojových kódů pak můžeme automatizovat například využitím CI popsaného v kapitole 4.6.2.

Produkt *ionCube* potřebný pro šifrování je tedy pro toto opatření nezbytný. IonCube je komeriční produkt, tzn. měl by se projevit ve výsledném nacenění. Výhodou zakoupení dále může být možnost zabránění spuštění kódu mimo potřebnou doménu či časové omezení funkčnosti.

[23] viz. http://php.net
[24] viz http://www.ioncube.com

Ukázku využití šifrovací funkce i šifrováných zdrojových kódů naleznete v elektronické příloze A.

Využitím tohoto opatření se může z aplikace pro potencionálního útočníka stát tzv.Black box, jež nemá možnost softwarovou část aplikace bez povšimnutí ovlivnit[25]. Co se týče změn dat v databázi, měla by být provedena bezpečnostní analýza, která data jsou zranitelná a zneužitelná a minimálně ty šifrovat.

[25]Pouze snad smazat. Případně ovlivnit-cracknout samotné PHP.

7 Závěr

V této práci jsem popsal dílčí části softwarového procesu tvorby informačních systémů, s kterými jsem se setkal v praxi. Během této mé praxe jsem začal klást vyšší důraz na fáze testování, specifikaci požadavků a business modelování, což byly části softwarového procesu, které významně pomohli zvednout kvalitu výsledného produktu. Zároveň jsem se mi podařilo vytvořit i systém, který jednotlivé části spojuje nebo podporuje.

Část zabezpečení aplikace byla přidána k této práci trochu uměle. Zde k většině zabezpečení přispívá výborný Nette Framework pana Davida Grudla a komunity Nette vývojářů. Nemalý přínos k bezpečnosti je i na straně Českého hostingu, který svými místy až paranoidními zabezpečeními k bezpečnosti webových aplikací přispívá. V sekci zabezpečení bych rád zdůraznil kapitolu o *Rizika spojená s interními útoky na aplikaci*, kde s použitím šifrovacích funkcí a šifrování zdrojových kódů můžeme z aplikace vytvořit šifrovanou pevnost, která lze prolomit snad jen prolomením rozšíření ionCube.

Automatické testování a CI stojí za zautomatizování a zefektivněním rutinních činností, které přímo zvyšují kvalitu výsledného produktu. Modulárnost aplikací zase podporuje možnosti testování tím, že aplikace je rozvrstvená, jednotlivé funkce jsou intiutivní a jdou testovat lépe než, tzv. spagetti code.

Prosím o pochopení, že cílem práce byl popis procesu, nikoli tvorba systému. V rámci této práce vzniklo několik teoretických návrhů podpořených implemetací, které jsou součástí přílohy.

7.1 Vývoj do budoucna

V rámci aplikace spec, by bylo přínosné, kdyby systém dokázal exportovat požadavky a analýzu do formy pdf. Hlavně z důvodu aby klient viděl systémovou složitost a práci, kterou je nutno pro vývoj systému vynaložit. Dále by mohl exportovat pouze změny stavů požadavků a analýzy mezi dvěma časovými údaji. Tyto údaje by byly vhodné pro ocenění aplikace.

V rámci procesu kontinuální integrace, by bylo vhodné zapojit nástroje, které měří kvalitu kódů (např. cyklomatická složitost aj.) a detekují softwarové nedostatky (Copy/Paste Detector aj.) jak popisuje [22].

8 Reference

[1] FORET, Miroslav. Marketingová komunikace. 2. aktualizované vydání. Brno : Computer Press, 2008. 451 s. ISBN 80-251-1041-9.

[2] LIPOVSKÁ, Lucie. *Multilevel marketing v 21. století.* [online] 2008, [cit. 1.5.2013]. Dostupné z: `http://dspace.k.utb.cz/handle/10563/5021`. Diplomová práce. Univerzita Tomáše Bati ve Zlíně.

[3] NOVOSAD, Antonín. *Multilevel marketing ve finančních službách* [online]. Zlín, 2010 [cit. 2013-02-14].
Dostupné z: `http://dspace.k.utb.cz/bitstream/handle/10563/12963/novosad_2010_bp.pdf`. Bakalářská práce. Univerzita Tomáše Bati ve Zlíně.

[4] *Řízení změn In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-14].*
Dostupné z: `http://cs.wikipedia.org/wiki/Řízení_změn`

[5] *Jak začít podnikat při studiu - Na volné noze, portál nezávislých profesionálů [online]. [cit. 2013-04-25].* `Dostupné z: http://navolnenoze.cz/blog/studenti/`

[6] *Cache. NETTE FOUNDATION. Dokumentace Nette Framework 2.1 [online]. [cit. 2013-02-14].*
Dostupné z: `http://doc.nette.org/cs/caching`

[7] ERDOGMUS, Hakan; MELNIK, Grigori; JEFFRIES, Ron. *Test-Driven Development.* 2011.
Dostupné z: `http://hakanerdogmus.net/weblog/wp-content/uploads/ESE-TDD-Chapter-Proofs-Final.pdf`

[8] KYJOVSKÝ, Tomáš. *Moderní nástroje pro zajištění kvality softwaru.* 2009.
Dostupné z: `http://is.muni.cz/th/72682/fi_m/thesis.pdf`

[9] *Apache HTTP server benchmarking tool* dokumentace Apache [cit. 2.5.2013]
Dostupné z: `http://httpd.apache.org/docs/2.2/programs/ab.html`

[10] FOWLER, Martin; FOEMMEL, Matthew. Continuous Integration - original. 2000. [cit. 1.4.2013]
Dostupné z: `http://www.martinfowler.com/articles/originalContinuousIntegration.html`

[11] FOWLER, Martin; FOEMMEL, Matthew. Continuous Integration. 2000. [cit. 1.4.2013]
Dostupné z: `http://www.martinfowlercom/articles/continuousIntegration.html`

[12] DUVALL P., MATYAS S. a GLOVER A. *Continuous Integration: Improving Software Quality and Reducing Risk* (2007, ISBN 0-321-33638-0)

[13] Kontinuální integrace s Jenkins CI, abclinuxu.cz, dostupné z: *http://www.abclinuxu.cz/blog/lojzoviny/2011/7/kontinualni-integrace-s-jenkins-ci*

[14] BERGMANN S., *PHPUnit Manual*, 2.4.2013, [cit. 6.5.2013], dostupné z: *http://www.phpunit.de/manual/current/en/phpunit-book.pdf*

[15] KRAVAL, Ilja. *Jaké jsou doporučené postupy při dělení systému na menší moduly.* Server objektových technologií [online]. 2009, roč. 2009, č. 67, s. 8 [cit. 2013-05-01]. Dostupné z: `http://www.objects.cz/clanky/clanek67/clanek67.pdf`

[16] *MVC aplikace & presentery. Nette Framework - Dokumentace: MVC aplikace & presentery [online]. 2008-2013 [cit. 2013-03-13].*

[17] CHACON S. *Pro Git.* Dostupné z: `http://knihy.nic.cz/`

[18] *Dokumentace nástroje Composer [cit. 2013-03-13] http://getcomposer.org/book.pdf*

[19] PHP: SQL Injection - Manual, 2001-2013 The PHP Group Dostupné z: `http://www.php.net/manual/en/security.database.sql-injection.php`

[20] *Zabezpečení před zranitelnostmi. Nette Framework - Dokumentace: Zabezpečení před zranitelnostmi [online]. 2008-2013 [cit. 2013-03-13].* `Dostupné z: http://doc.nette.org/cs/vulnerability-protection`

[21] *Šablony. Nette Framework - Dokumentace: Zabezpečení před zranitelnostmi [online]. 2008-2013 [cit. 2013-03-13].* `Dostupné z: http://doc.nette.org/cs/glossary#toc-escapovani`

[22] BERGMANN S. *Integrating PHP projects with Jenkins.* Farnham: O'Reilly, 2011. ISBN 978-144-9309-435.

[23] *API Nette Frameworku 2.x na serveru komunity NETTE* `http://api.nette.org/2.0/`

[24] *Dokumentace na serveru nástroje SELENIUM* `http://seleniumhq.org/docs/`

[25] *PHPunit dokumentace* `http://www.phpunit.de/manual/3.7/en/index.html`

[26] *Český hosting - Dokumentace: Zálohování dat [online]. 2013 [cit. 2013-03-14]. Dostupné z: http://www.cesky-hosting.cz/pro-zakazniky/napoveda/zalohovani-dat.html*

9 Seznam příloh

Přílohy v diplomové práci:

1. Příloha A: Uložené procedury pro traverzování

Elektronické přílohy:

1. příloha A: šifrování dat
2. příloha B: aplikace spec
3. příloha C: konfigurace závislostí
4. Příloha D: skeleton pro sběr požadavků
5. Příloha E: použití mezipaměti Cache
6. Příloha F: databázové závislosti
7. Příloha G: dotaz pro empirické ověření velikosti dotazu na databázi PostgreSQL

A Uložené procedury pro traverzování

V této kapitole jsou ukázány traverzovací procedury, které se spouštějí na straně databázového serveru *PostgreSQL* v jazyce *PL/PGSQL*.

Ve výpisu 2 je zobrazen zdrojový kód procedury getasc pro vyhledávání rodičovských id z tabulky uživatelé. Na obrázku a 3 je kód procedury *getdesc_fixedlimit* jež řeší traverzování grafem směrem k potomkům.

```
CREATE FUNCTION getasc(integer) RETURNS SETOF integer
    AS $_$
DECLARE
  idu_act INTEGER := $1;
  exist  INTEGER;
  user  RECORD;
BEGIN
  WHILE idu_act > -1 LOOP
    exist  := 0;
    FOR user IN (SELECT * FROM uzivatele WHERE idu=idu_act ) LOOP
      idu_act := user.idusponzor;
      exist  := 1;
      IF  idu_act <> -1 THEN
        RETURN NEXT idu_act;
      END IF;
    END LOOP;
    -- zabraneni zacykleni  pri   neexistenci  ID
    IF  exist  = 0 THEN
      RAISE EXCEPTION 'user␣with␣id␣%␣not␣found', idu_act;
    END IF;
  END LOOP;
END
$_$
LANGUAGE plpgsql;
```

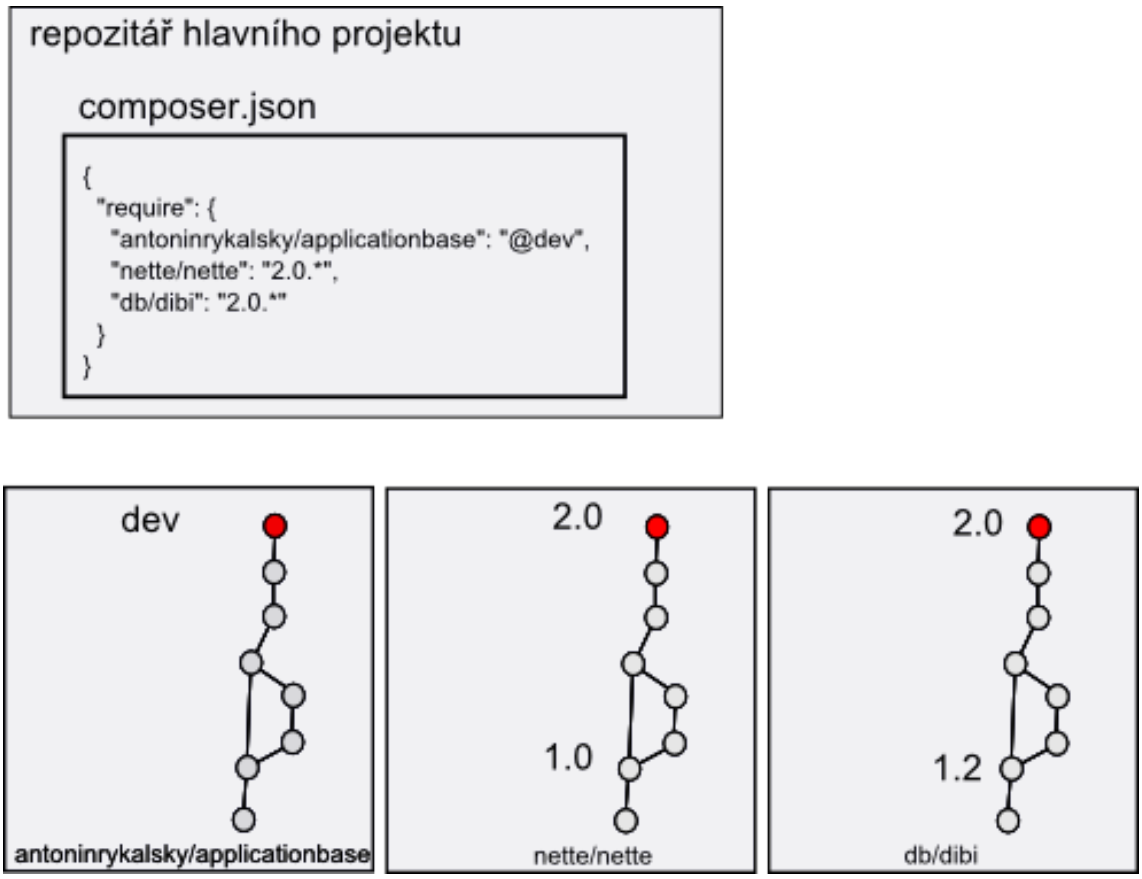
Výpis 2: Zdrojový kód procedury pro jednoduché traverzování grafem nahoru

```
-- zjisti  pocet potomku do ACT_LIMIT-té úrovně
DROP FUNCTION getdesc_fixedlimit(IN idu_this integer, IN act_limit integer,  IN  next_level
     integer,  OUT act_level integer,  OUT act_idu integer);
CREATE FUNCTION getdesc_fixedlimit(IN idu_this integer, IN act_limit integer,  IN  next_level
     integer,  OUT act_level integer,  OUT act_idu integer) RETURNS setof RECORD
    AS $_$
DECLARE
  idu_dalsi  INTEGER;
  depth INTEGER;
  prime RECORD;
  prime2 RECORD;
BEGIN

    -- pro vsechny prime potomky
    FOR prime IN (SELECT * FROM uzivatele WHERE idusponzor=idu_this ) LOOP
      act_idu= prime.idu;
      act_level  = next_level+1;
```

```
        RETURN NEXT;

          -- najdi neprime, predej vazbu na vetev

    IF next_level < 2 THEN
          FOR prime2 IN (SELECT * FROM getdesc_fixedlimit(prime.idu, 15, next_level )) LOOP
        act_idu = prime2.act_idu;
        act_level = prime2.act_level+1;
       RETURN NEXT;

          END LOOP;
          END IF;

     END LOOP;
END
$_$
LANGUAGE plpgsql;
```

Výpis 3: Zdrojový kód uložené procedury, která zajistí traverzování grafem nahoru s informací o zanoření ve struktuře

B Schématické zobrazení příkladu balíků a hlavního projektu pro Composer



Obrázek 12: Schématické znázornění